Plan repair driven by model-based agent diagnosis

Plan Repair Driven by Model-Based Agent Diagnosis

Roberto Micalizio
Dipartimento di Informatica
Università di Torino
corso Svizzera 185 - 10149 Torino, Italy

June 28, 2016

Abstract

Repairing a plan executed in a partially observable environment is a challenging task; especially when the plan to be repaired is part of a Multiagent Plan (MAP), and hence the synchronization among the agents further constrains the repair process.

The paper formalizes a local plan repair strategy, where each agent in a MAP is responsible for controlling (monitoring and diagnosing) the actions it executes, and for autonomously repairing its own plan whenever an action failure is detected.
The paper also describes how to mitigate the impact of an action failure on the plans of other agents when the local recovery strategy fails.

1 Introduction

Many real complex tasks find proper solutions in the adoption of a Multiagent Plan (MAP), in which a team of agents cooperate with one another to reach a common goal by performing actions concurrently.
The actual execution of a MAP, however, can be affected by the possible occurrence of *plan threats* (e.g., agent faults in [1]), that can disrupt the nominal progress of the plan by causing the failure of some actions. The occurrence of a plan threat does not prevent, in general, the agents from completing their activities, but the MAP needs to be repaired: a new planning process is required to overcome the effects of the occurred action failure, and achieve the global goal in some alternative way.

Dealing with action failures in a multi-agent setting is particularly challenging. First of all, since the agents cooperate by exchanging services, the local failure of an agent can easily propagate in the global MAP, that like in a domino effect, could start a series of harmful effects on the plan of other agents. Moreover, even though an impaired agent does not provide services to other agents, it may still represent a latent menace for them because it may lock critical resources indefinitely.

To cope with these issues, the paper proposes a local approach to autonomous plan

repair. In particular, the framework we propose enables each agent to perform a *closed control loop* over the actions in its local plan. This control loop includes three main tasks: *plan monitoring*, *agent diagnosis*, and *plan repair*. In this paper we discuss how these three activities can be realized and concatenated even when the system where the agents operate is just partially observable. The limited amount of observations represents a challenge: the monitoring can just estimate the agent state as a set of alternatives (i.e., a belief state), and the agent diagnosis is typically ambiguous (i.e., a set of alternative explanations); as a consequence the plan repair step must be able to deal with uncertain initial states and non-deterministic actions.

The paper is organized as follows: first, we introduce some basic notions on multi-agent plans; then we formalize the three steps of the control loop–monitoring, diagnosis, and repair. Since the repair strategy is based on a planning phase, we also sketch the conformant planner that we employ in our framework. Then we present some experimental results, and discuss some related works.

2 Related works

In [1] a model-based approach to plan diagnosis is presented, in this approach the authors relate the health status of a planning agent to the outcome of the planning activity. A similar approach has been adopted in this paper, however, we have explicitly considered a multiagent setting, in which the impaired agent tries to recover from an action failure without affecting the plans of other teammates.

The multi-agent setting is discussed in [15], where the authors introduce the notion of *plan diagnosis* as the subset of actions whose failure is consistent with the anomalous observed behavior of the system. In contrast to our work, this approach does not relate the failure of an action to the health status of the agents; it focus just on the detection of abnormal actions.

In [8, 9] the authors introduce the notion of *social diagnosis* to find the cause of coordination failures. In their approach, however, they do not explicitly consider plans, rather they model a hierarchy of *behaviors*: each agent selects independently from others the more appropriate behavior given its own beliefs.

The plan repair task has been addressed by a number of works (see e.g., [5, 6, 17]), which consider both self-interested and collaborative agents; however these works are not directly applicable in our framework.
These approaches in fact are mainly focused on repairing coordination flaws occurring during plan execution, thus they involve a re-scheduling task rather than performing a re-planning step (see the GPGP solution in [5]). In [6] a solution for reorganizing the tasks among the (collaborative) agents is presented: this approach is driven by the results of a diagnostic engine which explains detected plan failures. In this case, however, the explanations are derived from a causal model where anomalous events (e.g., resource unavailable) are organized in a fault tree, and the reaction to plan failure is a proper precompiled repairing solution.

3 Background

Multiagent Plans (MAPs). In this paper, a MAP is a system where a team $\mathcal{T}$ of agents actively cooperate for reaching a common goal G. For the sake of discussion, the model of a MAP is a simplified version of the formalism presented in [4] (which is an extension of the definition of a partial-order plan introduced in [18]). In particular, the MAP P is a tuple $\langle A, E, CL, RE \rangle$, where:

- A is the set of the action instances the agents have to execute. Two pseudo-actions, a_0 and a_∞, belong to A: a_0 (the starting action), has no preconditions, and its effects specify the propositions that are initially true; a_∞ (the ending action), has no effects, and its preconditions specify the propositions which must hold in the final state (i.e., the goal G of the MAP). Except for a_0 and a_∞, each action instance $a \in A$ is assigned to a specific agent $i \in \mathcal{T}$.

- E is a set of precedence links between actions: a precedence link $a \prec a'$ in E indicates that the execution of a must precede the execution of a'.

- CL is a set of causal links of the form $cl : a \xrightarrow{q} a'$; cl states that action a provides action a' with service q (q is an atom occurring in the preconditions of a').

- RE is a set of precedence links ruling the access to the resources. Since critical resources can only be accessed in mutual exclusion (see also the *concurrency requirement* in [15]), two actions a and a', assigned to different agents, cannot be executed at the same time instant if they require the same resource res. We assume that the planning process producing P also resolves the conflicts for accessing the resources by adding either $a \prec_{res} a'$ or $a' \prec_{res} a$. To keep a trace of these additional precedence links, they are labeled with the identifier of the specific resource they refer to, and are collected in the set RE.

Local Plans. The MAP P is decomposed into as many local plans as there are agents in the team: each local plan P^i is assigned to an agent i, and reaches a specific sub-goal G^i.
Formally, the local plan for agent i is the tuple
P^i=$\langle A^i, E^i, CL^i, T^i_{in}, T^i_{out}, RE^i_{in}, RE^i_{out} \rangle$ where A^i, E^i and CL^i have the same meaning of the sets A, E and CL, respectively, restricted to actions assigned to agent i. A^i includes also two special actions a^i_0 and a^i_∞ which specify, respectively, the initial and final conditions for the sub-plan P^i. T^i_{in} (T^i_{out}) is a set of incoming (outgoing) causal links of the form $a \xrightarrow{q} a'$ where a' (a) belongs to A^i and a (a') is assigned to another agent j in the team. Similarly, RE^i_{in} (RE^i_{out}) is a set of incoming (outgoing) precedence links of the form $a \prec_{res} a'$ where a' (a) belongs to A^i and a (a') is assigned to another agent j in the team.

We assume that each local plan P^i is a *totally ordered* sequence of actions: $P^i = \langle a^i_0, a^i_1, \ldots, a^i_\infty \rangle$.

Distributed plan execution. An agent executes its next action as soon as the action preconditions have been satisfied (the notions of preconditions and effects of an action will be formalized in the following section). However, an agent can execute no more

than one action in a given time instant. In particular, the time is assumed to be a discrete sequence of instants, and actions take one unit of time to be completed.
In the following the notation $a_l^i(t)$ will denote that the l-th action in the local plan P^i is executed by agent i at time t; when unnecessary, however, t will be omitted.
Coordination during plan execution. Since agents execute actions concurrently, they need to coordinate their activities in order to avoid the violation of the constraints defined during the planning phase. Effective coordination among agents is obtained by exploiting the causal and precedence links in the global MAP.
As pointed out in [5], coordination between two agents i and j is required when i provides j with a service q; this is modeled by a causal link $cl : a_h^i \xrightarrow{q} a_k^j$ in the MAP P. (As an effect of the MAP decomposition, cl belongs both to T_{out}^i and to T_{in}^j.) Since an agent can observe (at most) the direct effects of the actions it executes, only agent i has the chance of observing the achievement (or the absence) of q; thereby, the agent i must notify agent j about the outcome of action a_h^i.
Similarly, the consistent access to the resources is a form of coordination which involves precedence links. For example, the precedence link $pl : a_h^i \prec_{res} a_k^j$ means that agent i will release resource res to agent j just after the execution of action a_h^i; resource res will be used by agent j to execute action a_k^j. Of course, pl belongs both to RE_{out}^i and to RE_{in}^j.

Since the system is distributed, an agent does not have a global view of the status of all the system resources, but it knows just the status of the resources it holds. After having released resource res, agent i will not have access to the actual status of res. In the following we will denote as $AvRes(i, t)$ (available resources) the subset of resources assigned to agent i at time t; i.e., only agent i observes and knows the actual status of those resources.

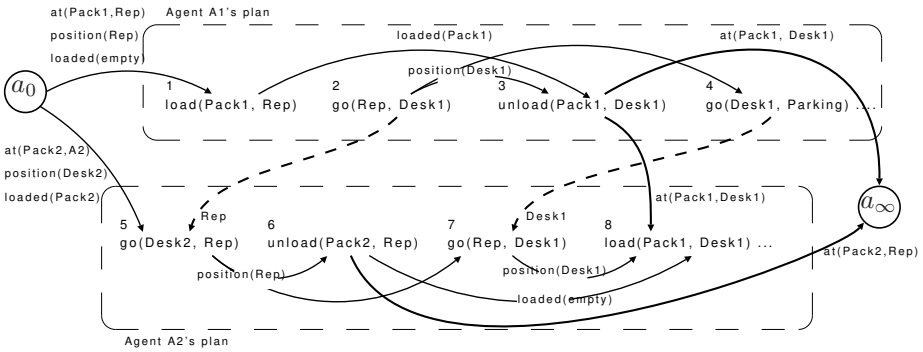

Figure 1: A simple example of MAP.

Example 1. Figure 1 shows a simple example of MAP we deal with. The domain of this MAP is an office-like environment similar to the ones presented in [10, 16]. In this domain, a team of agents has to deliver a number of packages to the employees' desks: an agent can load/unload a package from/to one desk, and carry one "heavy" package,

or two "light" packages around the office building. Critical resources are doors (connecting two rooms), desks, and repositories (special cabinets containing many packages). Critical resources can be used by only one agent per time; moreover, a desk can contain up to one package.

In the special case of Figure 1, the MAP involves two agents: `A1` and `A2`, whose local plans are enclosed within dashed rectangles. Pseudo-actions a_0 and a_∞ model, respectively, the initial state and the goal state. Local plans are totally ordered, so precedence links in E are not shown to simplify the picture. On the other hand, dashed edges between actions are used to represent precedence links in RE. For instance, the link from action `2` to action `5` states that action `5` requires resource `Rep`, and hence it can only start when action `2` terminates and releases `Rep`. Solid edges instead represent causal links; i.e., services that an action produces as a precondition for another action; in the picture, causal links are labeled with the provided service. For instance, the causal link from action `3` to action `8` states action `3` produces the service `at(Pack1, Desk1)`, which is one of the preconditions for action `8`. ■

4 Monitoring and Diagnosing a MAP

In this section we formalize the first two steps of the local control loop: the plan monitoring and the agent diagnosis activities. Before that, however, we first introduce some fundamental notions about the representation of the state of an agent and of the observations an agent collects during the plan execution phase.

Agent state. The state of agent i is modeled by a set of state variables VAR^i, partitioned into three subsets END^i, ENV^i and HLT^i. END^i and ENV^i denote the set of endogenous (e.g., the agent's *position*) and environment (e.g., the resources state) status variables, respectively. Because of the partitioning, each agent i maintains a private copy of the resource status variables; therefore for each resource $res_k \in RES$ ($k : 1..|RES|$) the private variable $res_{k,i}$ is included in the set ENV^i. The consistency among all these copies is assured by the fact that conflicts for accessing the resources are solved at planning level. The precedence links in RE guarantee in fact that, at each time t, a resource res_k is available just for an agent i (i.e., res_k belongs to $AvRes(i, t)$); therefore for any other agent $j \in \mathcal{T} \setminus \{i\}$ the status of the res_k is *unknown*.

Since we are interested in monitoring the plan execution even when something goes wrong, we introduce a further set HLT^i of variables for modeling the health status of an agent. For each agent functionality f, a variable $v_f \in HLT^i$ represents the health status of f. Generally speaking, the domain of variable v_f is a set $\{ok, abn_1, \ldots, abn_n\}$ where ok denotes the nominal mode, while $abn_1, \ldots, abn_n$ denote anomalous or degraded modes. An action failure can be therefore explained in terms of faults in a subset of functionalities of a specific agent.

System observability. We assume that after the execution of an action $a_l^i(t)$ the agent i receives a set $obs^i(t + 1)$ of observations, that conveys information about a subset of variables in VAR^i. Given the partial observability, an agent can directly observe just the state of its available resources, and the value of a subset of variables in END^i,

Table 1: The extended model of action a_2:`go(Rep,Desk1)` (a simplified version).

		active variables at time t				*active variables at time t+1*			
		pos	**loaded**	**pwr**	**engTmp**	**pos**	**loaded**	**pwr**	**engTmp**
1	*nominal*	`Rep`	*empty*	*ok*	*ok*	`Desk1`	*empty*	*ok*	*ok*
2	*nominal*	`Rep`	*obj*	*ok*	*ok*	`Desk1`	*obj*	*ok*	*ok*
3	*degraded*	`Rep`	*empty*	*low*	*ok*	`Desk1`	*empty*	*low*	*ok*
4	*degraded*	`Rep`	*empty*	*ok*	*hot*	`Desk1`	*empty*	*ok*	*hot*
5	*faulty*	`Rep`	*obj*	*low*	*ok*	`Rep`	*obj*	*low*	*ok*
6	*faulty*	`Rep`	*obj*	*ok*	*hot*	`Rep`	*obj*	*ok*	*hot*
7	*faulty*	`Rep`	*obj*	*ok*	*ok*	`Rep`	*obj*	*low*	*ok*
8	*faulty*	`Rep`	*obj*	*ok*	*ok*	`Desk1`	*obj*	*low*	*ok*

whereas the variables in HLT^i are not directly observable and their actual value can be just inferred. As a consequence, at each time t the agent i can just estimate a *set* of alternative states which are consistent with the received observations; in literature this set is known as *belief state*, and in the following the notation $\mathcal{B}^i(t)$ will refer to the belief of agent i inferred at time t.

Action models. In order to monitor the execution of action $a_l^i(t)$, agent i needs a model for estimating all the possible, nominal as well as anomalous, evolutions of the action itself. In our framework, an action model is a tuple $\langle var(a_l^i), \Delta(a_l^i(t))\rangle$, where:

- $var(a_l^i(t)) \subseteq VAR^i$ is the subset of *active* state variables; namely, the variables over which the preconditions and the effects of the action are defined.
- $\Delta(a_l^i) \subseteq var(a_l^i(t)) \times var(a_l^i(t))$ is a transition relation defined over the active variable, and representing the set of possible evolutions in the agent state when the agent performs a_l^i. More precisely, $\Delta(a_l^i)$ can be thought of as a set of tuples of the form $\langle s_t, s_{t+1}\rangle$, where s_t is a (partial) agent state at time t (when the action starts) and s_{t+1} is another (partial) agent state at time t+1 (when the action ends).

For brevity, in the following we will denote as $PRE(a_l^i)$ (i.e., preconditions) the projection of $\Delta(a_l^i)$ over the state variables at time t, and $EFF(a_l^i(t))$ (i.e., effects) the projection of $\Delta(a_l^i)$ over the state variables at time t+1.

Example 2. Table 1 reports a simplified model for the `go` action in our office domain. The action requires an agent to move from its current location to another one. For simplicity, Table 1 shows the instantiated model of a `go` action, rather than a template, in which the initial location is repository `Rep`, whereas desk `Desk1` is the final location. The outcome of a `go` action depends on the health state of a subset of functionalities; in the specific case: `pwr` represents the available power (i.e., battery charge) which can either be *ok* or *low*; similarly, `engTmp` models the temperature of the engine which is either nominal *ok*, or abnormal *hot*. Finally, it is worth noting that the outcome of a `go` action can also be affected by contextual conditions, as for instance the presence

of a block on board the agent, encoded by variable `loaded`. All these variables are therefore taken into account in the definition of the `go` action model for the monitoring purpose. More precisely, each entry of such a relation represents a possible state transition defined over the active variables. Note that the first four entries and the last one represent situations in which the action achieves its expected effects. However, only the first two entries are not affected by faults. The other entries represent situations in which the nominal expected effects are not achieved. ■

Given action a_l^i, *healthVar*(a_l^i)=$HLT^i \cap var(a_l^i)$ denotes the set of variables representing the health status of the functionalities which directly affect the outcome of action a_l^i. These variables are fundamental to infer the expected, nominal effects of a_l^i given its model $\Delta(a_l^i)$. First, we define the healthy formula *healthy*(a_l^i) of action a_l^i as the restriction of each variable $v \in$ *healthVar*(a_l^i) to the nominal behavioral mode *ok*. It is easy to see that the healthy formula of an action represents the nominal health status required to agent i for successfully completing the action itself. Then the nominal effects are defined as follows.

Definition 1 *The set of the nominal effects of action a_l^i is nomEff(a_l^i)=*$\{q \in EFF(a_l^i) \setminus HLT^i \mid PRE(a_l^i) \cup$ *healthy*$(a_l^i) \models q\}$.

Intuitively, the nominal effects of action a_l^i are only achieved when all the functionalities mentioned in $healthVar(a_l^i)$ are in their nominal mode ok. Note that the health variables are not part of the nominal effects; indeed, these variables are not even observable.See for instance transitions 1 and 2 of the action model in Table 1. On the other hand, when at least one variable $v \in$ *healthVar*(a_l^i) assumes an anomalous mode (i.e., a functionality is not in the nominal mode), the behavior of the action may be non-deterministic and some of the expected effects may be missing. For instance, transitions 7 and 8 of the action model in Table 1 share the same preconditions, but evolve differently due to the occurrence of a fault affecting the power functionality: in the first case the agent is unable to move from its initial position `Rep`, in the second one, instead, the agent gets in position `Desk1`.

In the following, $\mathcal{A}$ will denote the set of action models an agent exploits for monitoring the progress of its own plan.

The estimation of the agent status. The estimation process aims at predicting the state of agent i at time $t+1$ after the execution of an action a_l^i(t). However, because of the non determinism in the action models, and the partial system observability, the estimation process can in general infer just a set of alternative agent states (i.e.; a belief state) rather than the actual agent state. The estimation can be formalized in terms of the relational algebra operators as follows.

Definition 2 *Let $\mathcal{B}^i(t)$ be the belief state of agent i, let $\Delta(a_l^i(t))$ be the model of the action executed at time t, the agent belief state at time $t+1$ is:*

$\mathcal{B}^i(t+1)$ =

$\text{PROJECTION}_{t+1}[$

$\text{SELECTION}_{obs^i(t+1)}$ $(\mathcal{B}^i(t)$ JOIN $\Delta(a_l^i(t)))]$

The (natural) join operation $\mathcal{B}^i(t)$ JOIN $\Delta(a_l^i(t))$ is the predictive step by means of which all the possible agent states at time $t+1$ are estimated. The selection $\text{SELECTION}_{obs^i(t+1)}$, refines the predictions by pruning off all those estimates which are

inconsistent with the agent observations. Finally, the belief state $\mathcal{B}^i(t+1)$ is obtained by projecting the resulting estimates over the status variables of agent i at time $t+1$.

Action outcome. The outcome of an action is either *succeeded* or *failed*. However, since agent belief states are typically ambiguous, determining the outcome of an action might be not easy. In this paper we adopt a *strong committed* policy, and assume that as soon as an action has been completed with success, the agent receives an amount of observations which is sufficient to detect the achievement of all action nominal effects. More precisely, given a belief $\mathcal{B}^i(t+1)$, agent i determines the successful completion of action $a_l^i(t)$ as follows:

Definition 3 *The outcome of action $a_l^i(t)$, is* succeeded *iff $\forall q \in nomEff(a_l^i(t))$, $\forall s \in \mathcal{B}^i(t+1)$, $s \models q$.*

Thus, action $a_l^i(t)$ is successfully completed only when all atoms q in $nomEff(a_l^i(t))$ are satisfied in every state s in $\mathcal{B}^i(t+1)$: all the nominal effects of $a_l^i(t)$ hold in every possible state estimated after the execution of the action. When we cannot assert that action $a_l^i(t)$ is succeeded, we assume that the action is failed. This conservative assumption can be released along the line discussed in [11, 12], in which a *weak committed* policy is introduced, allowing an action outcome to be *pending* when the available observations are not sufficient for discriminating between success and failure.

Agent diagnosis. Whenever an agent i detects the failure of one of its action, say $a_l^i(t)$, the agent starts a diagnostic procedure. In principle, an agent belief state $\mathcal{B}^i(t+1)$ is already a diagnosis for the detected failure of action $a_l^i(t)$: $\mathcal{B}^i(t+1)$ contains all the possible states of agent i consistent with the observations gathered by i, and each of these states is a possible explanation for the failure. However, an agent belief state is in general highly ambiguous, and it is not really useful to identify the root causes of the action failure. Indeed, our objective is to understand why an action has failed, and try to recover from this failure by overcoming its root causes. This means that the agent belief state needs to be refined in order to become a useful source of diagnostic information.

As we have already noticed in Definition 1, an action a_l^i achieves its expected, nominal effects only when the healthy formula $healthy(a_l^i)$ holds. Thus, since we have detected the failure of a_l^i, the healthy formula must not hold; that is, at least one functionality of agent i is not properly working.

Therefore, an explanation expressed in terms of the status variables in $healthVar(a_l^i)$, rather than all the agent state variables, is sufficient to identify the root causes of the action failure.

More formally, by using the operators of the relational algebra, an agent diagnosis is inferred as

$$D^i = \text{PROJECTION}_{healthVar(a_l^i)} \mathcal{B}^i(t+1)$$

An agent diagnosis is therefore a relation defined over the variables in $healthVar(a_l^i)$, where each entry represents a possible explanation for the failure. Of course, D^i can still be ambiguous, but its cardinality is usually negligible if compared with the cardinality of the original belief state.

Example 3. Let us consider a simple scenario in which after the execution of action a_2 :`go(Rep, Desk1)` of our running example, agent `A1` receives as observation the atom pos =`Rep`. From the action model exemplified in Table 1, it is easy to conclude that action a_2 has not reached its expected effects. In fact, only transitions `5`, `6`, and `7` are consistent with two elements: 1) the current position of `A1` is still `Rep` (just observed); and 2) the agent is currently loaded with an object (the previous action was a load, which has been completed successfully, otherwise the agent would have stopped due to the strong committed policy). Agent `A1` therefore infers a diagnosis by projecting the belief state estimated with the three transitions mentioned above over the health variables in $healthVar(a_2)$. The result is the following set of possible explanations:

$$D^{\mathtt{A1}} = \{\langle \mathbf{pwr} : low,\ \mathbf{engTmp} : ok\rangle \\ \langle \mathbf{pwr} : ok,\ \mathbf{engTmp} : high\rangle\}$$

The first explanation blames a low charge in the battery as a root cause of the failure of action a_2. Indeed, the action model for the `go` action states that when an agent is loaded with a block (either light or heavy), and its battery charge is low, then the agent cannot move. Similarly, the second explanation blames a high temperature in the engine as the root cause of the failure; also in this case, in fact, an agent cannot move if it is loaded with an object. It is worth noting that, even though the `go` action cannot be completed under these contextual conditions, other actions could be performed. For instance, an `unload` action can be performed even though the battery charge is low or the engine temperature is high. This is essential for the plan repair purpose, as we will see in the next section. ■

Missing Goals. As noted earlier, the agents in the team $\mathcal{T}$ cooperate one another by exchanging services; that is, there exist causal dependencies between actions of different agents. As a consequence, the failure of action a_l^i prevents the execution of the actions in the plan segment $[a_{l+1}^i, a_\infty^i]$ and, since some services will never be provided, it can indirectly impact the local plans of the other teammates.
The set of the services that agent i can no longer provide due to the failure is denoted as the set of *missing goals*; singling out these services is important as, in principle, it would be sufficient to find an alternative way to provide them in order to reach the global goal G despite the failure of action $a_l^i(t)$.

Definition 4 *Given the failure of action a_l^i,*
let $[a_{l+1}^i, \ldots, a_\infty^i]$ be the plan segment the agent i is unable to complete, the set of missing goals is: $\mathcal{MG}(i)$=\{ service q|
$\forall a_k^i \in [a_l^i, \ldots, a_\infty^i]$, $q \in nomEff(a_k^i)$, *and*
$q \in PRE(a_\infty^i)$ *or*
$\exists$ *a causal link* $cl \in CL$ *such that*
$cl : a_k^i \xrightarrow{q} a_h^j; i \neq j)\}$

Namely, the service q is a missing goal when q is a nominal effect no longer provided by an action in the plan segment $[a_l^i, \ldots, a_\infty^i]$, and when either q is an atom appearing in the sub-goal G^i (i.e., q is a precondition of the special action a_∞^i) or q is a service agent i should provide to another agent j.

5 Plan Repair: a local strategy

In this section we discuss a methodology for repairing the local plan P^i, interrupted after the failure of action a_l^i has been detected. Essentially, this repairing process consists in a re-planning step intended to overcome, if possible, the harmful effects of the failure. In the next section we sketch the planning algorithm activated by agent i; in this section we focus on which goals should be reached for the recovering purpose. As noted above, the set of missing goals can be used to this end; unfortunately, when the recovery is driven by the missing goals it requires global changes; in fact, the missing goals are long term objectives, that can be reached by acquiring new resources; the acquisition of a resource, however, imposes the coordination with other teammates, and hence new causal and precedence links are to be introduced in the global MAP P; it follows that a number of other agents in the team have to change their local plans.

The local strategy we propose, instead, tries to recover from the failure of a_l^i just by changing the local plan P^i, without any direct impact on the plans of other teammates. The idea of a local strategy stems from the observation that in many cases an agent is still able to do something useful even if its health status is not completely nominal. By exploiting this possibility, we first formalize a local replanning strategy intended to overcome the causes of an action failure, and then restore the plan execution from the point where it was stopped. However, when such a replanning step fails, we also show how the agent in trouble can reduce the impact of the failure in the MAP P by moving into a safe status.

Repairing the interrupted local plan. This step is based on the observation that the plan segment $[a_{l+1}^i, \ldots, a_\infty^i]$ could be carried out if the root causes of the failure of action a_l^i were removed.

These root causes have been singled out by the diagnostic inferences: the agent diagnosis D^i explains the failure of a_l^i as a combination of anomalous conditions in the functionalities of agent i; therefore, to overcome the causes of this failure the agent i has to restore a healthy condition in those functionalities. To this end the agent i can exploit a set $\mathcal{A}^R$ of *repairing actions*, each of which restores the healthy condition in a specific agent functionality. Of course, it is possible that some faults cannot be autonomously repaired by an agent. For example, in our office domain, a robotic agent can autonomously fix a low charge in the battery by means of a `recharge` repairing action; whereas a fault in the mobility functionality cannot be fixed by the agent autonomously, and requires an external (possibly human) intervention.

Therefore, relying on the agent diagnosis D^i, agent i assesses whether one (or a set of) repair action(s) exists. If recover actions do not exist, agent i gives up the synthesis of a recovery plan and tries to reach a *safe status* (see later). If recovery actions exist, the agent i tries to reach a new goal $\mathcal{K}$ consisting in: 1) restoring the healthy conditions in its functionalities by executing an appropriate set of repairing actions; and 2) restarting the execution of the plan from the failed action a_l^i. A *repairing plan* Pr^i is a plan which meets these two goals, and it can be found by resolving the following planning problem:

Definition 5 *The repairing plan* Pr^i=$[ar_0^i, \ldots, ar_\infty^i]$ *is a solution of the planning problem*

$\langle \mathcal{I}, \mathcal{F}, \overline{\mathcal{A}}, AvRes(i,t), D^i \rangle$*; where:*

- $\mathcal{I}$ *(initial state) corresponds to the agent belief state:* $EFF(ar_0^i) \equiv \mathcal{B}^i(t+1)$ *(i.e., the belief state inferred after the execution of action* $a_l^i(t)$*).*
- $\mathcal{F}$ *(final state) is the goal* $\mathcal{K}$ *defined as* $PRE(ar_\infty^i) \equiv \{\forall v \in healthVar(a_l^i),\ v = ok\} \wedge PRE(a_l^i)$
- $\overline{\mathcal{A}} \subseteq \mathcal{A} \cup \mathcal{A}^R$ *is the set of action models which can be used during the planning process given the agent diagnosis* D^i*, and the available resources* $AvRes(i,t)$*.*

A repairing plan Pr^i must also satisfy two demanding requirements:

Requirement 1 *Since the repairing plan* Pr^i *can impose local changes only, no new resources can be acquired: the actions in* Pr^i *can just exploit the resources in* $AvRes(i,t)$*, already acquired by agent* i *at the time of the failure.*

Requirement 2 *Since the belief state* $\mathcal{B}^i(t+1)$ *is potentially ambiguous (the actual agent health status is not precisely known) the repairing plan* Pr^i *must be conformant, namely, it must be executable no matter the actual health status of agent* i*.*

An important consequence of the conformant requirement is the following property.

Property 1 *For each action* $ar_k^i \in Pr^i$ *it must hold* $healthy(ar_k^i) \cup D^i \nvdash \bot$

Property 1 states that all the actions in the repairing plan must be executable despite the current status of agent i is not healthy. Therefore, when no action is executable given the agent diagnosis D^i, a repairing plan does not exist.

In some cases an agent diagnosis could explain an action failure both with faults that are repairable, and with faults that are not. For instance, the failure of a $\mathtt{go}$ action could be explained either as a low battery level (repairable), or as a broken engine (not repairable). In such a situation, any attempt to find a conformant plan that restores the nominal behaviors in both the battery and the engine will fail as there is no way, for an agent, to self-repair its own engine. To deal with agent diagnoses as such, we propose a preference criterion based on the repairability of faults. More precisely, we keep an explanation exp in D^i as far as all the faults assumed in exp are repairable; exp is pruned off, otherwise. Of course, if after this refinement, the agent diagnosis gets empty, then there is no way to recover from the action failure (all explanations include at least one non-repairable fault), and hence the repairing procedure is not even attempted.

Assuming that the plan Pr^i exists, agent i yields its new local plan $P^{*i} = [ar_0^i, \ldots, ar_\infty^i] \circ [a_l^i, \ldots, a_\infty^i]$; where $\circ$ denotes the concatenation between two plans (i.e., the second plan can be executed just after the last action of the first plan has been executed).

Property 2 *The recovery plan* P^{*i} *is feasible and executable.*

Due to space reason the proof is omitted, intuitively the feasibility of the recovery plan P^{*i} stems by two characteristics: 1) every plan segment is feasible on its own as it has been produced by a specific planning step, and 2) the preconditions of the action ar_∞^i of the first plan matches with the effects of the action a_l^i of the second one. A more important property is the following one:

Property 3 *The recovery plan P^{*i} meets all the services in $\mathcal{MG}(i)$.*

Property 3 guarantees that, by executing P^{*i} in lieu of $[a_l^i, \ldots, a_\infty^i]$, agent i can recover from the failure of action a_l^i and achieve its sub-goal G^i despite the failure.

Reaching the Safe Status. The repairing plan Pr^i, however, may not exist. In fact, the faults assumed in D^i may be not repairable, or a conformant solution may not exist. When the plan recovery process fails, the impaired agent can be seen as a latent menace for the other team members (e.g., when the agent locks indefinitely critical resources). We complement the first step of the local strategy by means of a further step intended to lead the impaired agent i into a *safe status* $\mathcal{S}^i$. In this paper, we define a safe status as a condition where all the resources used by i at time t–the time of the failure of action $a_l^i(t)$–have been released. Also this step can be modeled as a planning problem as follows:

Definition 6 *The plan-to-safe-status Ps^i=$[as_0^i, \ldots, as_\infty^i]$ is a solution of the planning problem*
$\langle \mathcal{I}, \mathcal{F}, \overline{\mathcal{A}}, AvRes(i,t), D^i \rangle$; where:

- *$\mathcal{I}$ (initial state) corresponds to the agent belief state: $EFF(as_0^i) \equiv \mathcal{B}^i(t+1)$ (i.e., the belief state inferred after the execution of action $a_l^i(t)$).*
- *$\mathcal{F}$ (final state) is the safe status $\mathcal{S}^i$, defined as $PRE(as_\infty^i) \equiv \forall res_k \in AvRes(i,t)$, $res_{k,i}$ = free.*
- *$\overline{\mathcal{A}}$ is as before the set of action models which can be used during the planning process given the agent diagnosis D^i and the set of available resources $AvRes(i,t)$.*

Of course, also the plan-to-safe-status Ps^i must satisfy the requirements 1 and 2; thus property 1 can be extended to the actions in Ps^i too. Repairing actions can be used also during this planning step, in some cases in fact it is required to restore the healthy status in some functionalities to release a resource.

When the plan-to-safe-status Ps^i exists, it becomes the new local plan assigned to agent i; that is $P^{*i} = Ps^i$, and all the actions in $[a_l^i, \ldots, a_\infty^i]$ are aborted. Therefore, even though agent i is unable to reach its goal G^i, it moves into safe status in order to do not hinder the other team members in their activities.

When the recovery strategy fails. The recovery strategy fails when neither the plan to a repaired state, nor the plan to a safe status exists. In this case, we adopt a conservative policy and we impose that the impaired agent gives up the execution of its local plan. Performing further actions, in fact, may lead the agent in dangerous conditions; for example, the agent could lock indefinitely some resources preventing others to access them.

The failure of the local recovery strategy does not imply, in general, that the action failure cannot be recovered from, but different, global strategies should be activated. These strategies (out the scope of this paper), are driven by the set of missing goals, and may require the cooperation of a subset of agents, or the activation of a global re-planning step.

The algorithm. The high-level algorithm of the control loop performed by each

LocalControLoop$(P^i, \mathcal{B}^i(0))$

1. $t \leftarrow 0$
2. **while** there are actions in P^i to be executed **do**
3. $a_l^i \leftarrow$ **nextAction**(P^i)
4. **if** $PRE(a_l^i)$ are satisfied in $\mathcal{B}^i(t)$ **then**
5. EXECUTE a_l^i
6. **gather observations** $obs^i(t+1)$
7. $\mathcal{B}^i(t+1) \leftarrow$ **Monitoring**$(\mathcal{B}^i(t), \Delta(a_l^i))$
8. **if outcome**$(a_l^i, \mathcal{B}^i(t+1))$ equals *failed* **then**
9. $D^i \leftarrow$ **Infer-Diagnosis**$(\mathcal{B}^i(t+1), \mathit{healthVar}(a_l^i))$
10. $Pr^i \leftarrow$ **ConfPlan**$(\mathcal{B}^i(t+1), \mathcal{K}, \mathcal{A}, AvRes(i,t), D^i)$
11. **if** Pr^i is not empty **then**
12. $P^i \leftarrow Pr^i \circ [a_l^i, .., a_\infty^i]$
13. **else**
14. $Ps^i \leftarrow$ **ConfPlan**$(\mathcal{B}^i(t), \mathcal{S}, \mathcal{A}, AvRes(i,t), D^i)$
15. **if** (Ps^i is not empty) **then**
16. $P^i \leftarrow Ps^i$
17. **else**
18. invoke a global recovery strategy
19. **end if**
20. **end if**
21. **end if**
22. $t \leftarrow t+1$
23. **end if**
24. **end while**

Figure 2: The control loop algorithm.

agent $i \in \mathcal{T}$ is showed in Figure 2. The algorithm consists in a while loop, where at each iteration agent i singles out the next action a_l^i to be executed. The action a_l^i is executed iff its preconditions are satisfied in the current belief state $\mathcal{B}^i(t)$. After the action execution, i gathers the available observations and detects the outcome of a_l^i (see Definition 3). In case the action outcome is *failed*, first the diagnostic inference are activated, and then the conformant planner is invoked to find a plan to a repaired state (Definition 5), or alternatively a plan to safe status (Definition 6). When both the planning steps fail, the agent i sends a message to all the other agents about its failure, and interrupts the execution of its local plan P^i.

6 Conformant Planning

In this section we just sketch the main steps performed by the conformat planner we propose, more details can be found in [10].

The high-level algorithm of the conformant planner we propose is showed in Figure 3. The simple idea at the basis of our planner is to create a macro-operator Φ which

ConfPlan$(\mathcal{I}, \mathcal{F}, \mathcal{A}, AvRes(i,t), D^i)$
1. $\Phi \leftarrow$ **Build-**$\Phi(\mathcal{A}, AvRes(i,t), D^i)$
2. $\mathcal{PSET}_0 \leftarrow \mathcal{I}$
3. $h \leftarrow 0$
4. *solved*←*false*
5. **while** *not solved and* $h < MAXDEPTH$ **do**
6. $\quad \mathcal{PSET}_{h+1} \leftarrow \mathcal{PSET}_h$ JOIN Φ
7. $\quad \mathcal{PSET}_{h+1} \leftarrow$ **PruneNotConformant**$(\mathcal{PSET}_h, \mathcal{PSET}_{h+1})$
8. $\quad$ **if** $\mathcal{PSET}_{h+1}$ is *empty* **then**
9. $\quad\quad$ **return** $\emptyset$
10. $\quad$ **end if**
11. $\quad solved \leftarrow$ **CheckGoal**$(\mathcal{PSET}_{h+1}, \mathcal{F})$
12. $\quad$ **if** $solved$ equals *true* **then**
13. $\quad\quad \pi \leftarrow$**ExtractPlan**$(\mathcal{PSET}_{h+1})$
14. $\quad$ **else**
15. $\quad\quad h \leftarrow h + 1$
16. $\quad$ **end if**
17. **end while**
18. **return** π

Figure 3: The high-level algorithm for the synthesis of a conformant plan.

gathers, in a disjunctive form, all the models of the actions that can be used during the plan repair phase. This task is performed by function **Build**-Φ (line 1), which takes in input the set of action models $\mathcal{A}$, the set of available resources $AvRes(i,t)$, and the (preferred) agent diagnosis D^i. The function selects from $\mathcal{A}$ all those actions that can be performed given the set of resources currently assigned to agent i, and that are consistent with the diagnosis D^i. In other words, an action that requires a functionality assumed faulty in D^i, or that requires a resource that the agent does not hold, is not included in Φ.

The macro-operator Φ is subsequently used to extend the set of current plan hypotheses. To keep a trace of such plan hypotheses, we introduce a further structure, named $\mathcal{PSET}$.

Intuitively, $\mathcal{PSET}_h$ can be seen as a set of trajectories, where each trajectory tr has the form

$$\langle \mathcal{B}^i(0), a_1, \mathcal{B}^i(1), a_2, \ldots, a_h, \mathcal{B}^i(h) \rangle.$$

Each $\mathcal{B}^i(k)$ $(k : 0..h)$ represents an agent belief state, while each a_k $(k : 1..h)$ represents an action that brings belief state $\mathcal{B}^i(k-1)$ to evolve into belief state $\mathcal{B}^i(k)$. In particular, we call these actions "conformant" in the sense that each action a_k is applicable in every state $s \in \mathcal{B}^i(k-1)$. In other words, $\mathcal{PSET}_h$ represents the set of conformant plans of length h found so far. At the beginning of the algorithm, of course, $\mathcal{PSET}_0$ is set to the initial belief state $\mathcal{I}$ (line 2).

After these preliminary steps, the algorithm starts a while loop that terminates either when a conformant plan has been found, or when a maximum depth $MAXDEPTH$ has been reached. In this second case, introduced to guarantee the termination of the algo-

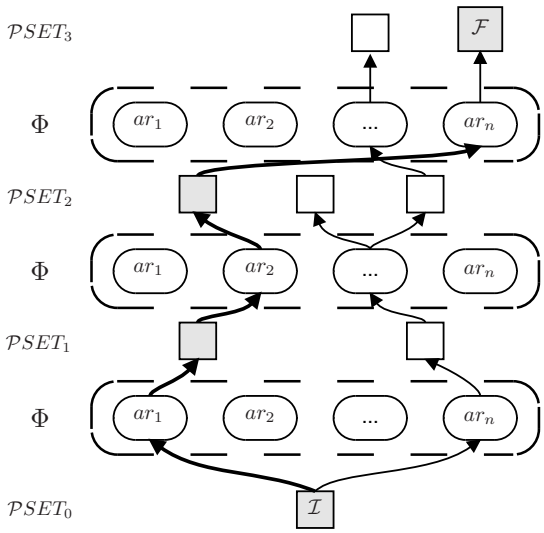

Figure 4: An example of how the $\mathcal{PSET}$ structure is extended.

rithm, the algorithm has explored all the space of plans no longer than $MAXDEPTH$ actions without finding a solution, and hence terminates with a failure.

At each iteration of the while loop, the current $\mathcal{PSET}$ structure is extended by means of the macro-operator Φ (line 6). Such an extension consists in applying each action in Φ to each belief state in the frontier of $\mathcal{PSET}_h$; namely, to each belief state at depth h. The result of this operation is a new set of plan hypotheses $\mathcal{PSET}_{h+1}$, which are an action longer than the previous ones. Note that this extension is carried out by means of a natural JOIN, which does not guarantee that $\mathcal{PSET}_{h+1}$ contains conformat actions only. For this reason, the newly created $\mathcal{PSET}_{h+1}$ is refined by removing all those belief states produced by non-conformant actions (line 7). If the resulting $\mathcal{PSET}_{h+1}$ gets empty, the planning process terminates with the guarantee that no conformat plan exists for the given problem. Otherwise, the algorithm checks whether some of the belief states in the frontier of $\mathcal{PSET}_{h+1}$ satisfy the goal. In the positive case, a conformant plan has been found, so it is extracted and returned as a result. In the negative case, the loop is repeated.

Figure 4 gives an intuition of how the search proceeds. Belief states are represented as boxes. Grey boxes are those along a solution. The macro-operator Φ contains all the actions that can be used to find a conformant plan (i.e., $ar_1, ar_2 \ldots ar_n$). The operator is applied to each belief state in the frontier of the current $\mathcal{PSET}$ structure starting from the initial belief $\mathcal{I}$. After three steps of the algorithm, a solution is found since a belief state satisfies the goal $\mathcal{F}$, and hence the plan $\langle ar_1, ar_2, ar_n \rangle$ is returned.

The planning algorithm **ConfPlan** has some interesting properties.

Property 4 (Optimality) *If* **ConfPlan** *terminates with success, it finds all the optimal (i.e., with the minimum number of actions) conformant plans.*

Intuitively, the proof follows by the fact that the algorithm implements an exhaustive,

forward-chaining search. Thus, it is possible to demonstrate that if a conformant plan π found by **ConfPlan** is longer than another conformant plan π', for the same problem, then π' must mention at least one action that is not included in the macro-operator Φ, but in this case, π' would not be executable by the impaired agent i.

Property 5 (Soundness) *If* **ConfPlan** *terminates with a failure because the set $\mathcal{PSET}$ gets empty, then there not exists a conformant plan to solve the problem.*

Also this property follows by the fact that the search is exhaustive: if the $\mathcal{PSET}_{h+1}$ gets empty when $h+1 \leq MAXDEPTH$, it means that no action in Φ is applicable as a conformant action to any of the belief states in the frontier of $\mathcal{PSET}_h$. Therefore, no solutions longer than h actions are possible.

Property 6 (Incompleteness) *If* **ConfPlan** *terminates with a failure because the threshold $MAXDEPTH$ has been reached, a conformant plan longer than $MAXDEPTH$ actions might exist.*

The incompleteness of the **ConfPlan** algorithm comes from the fact that the search is made in a forward chaining manner without backtracking. Thus, to guarantee the termination even in presence of infinite paths, we need to set an artificial limit to the search space, which introduces the incompleteness.

Example 4. Let us consider again the situation depicted in Example 4. In particular, let us consider the agent diagnosis D^i inferred after the failure of action a_2. Relying on such a diagnosis, agent `A1` starts a plan repair phase. First of all, the agent checks whether the faults assumed by the agent diagnosis are repairable or not; i.e., whether there exists an action that restores the nominal behavioral mode in the functionality affected by those faults. In our specific case, the two faults are both repairable. Thereby, agent `A1` proceeds by building the Φ macro-operator by considering the resources it currently holds $AvRes(\texttt{A1}, t) = \{\texttt{Rep}, \texttt{Desk1}, \texttt{Pack1}\}$ (where t is the current time); in addition, `A1` has also access to the `Parking` area, which is not a resource since it is not constrained. Therefore, the set $\mathcal{A}$ of actions that `A1` can perform is:

- `load(Rep, Pack1), unload(Rep, Pack1)`
- `load(Desk1, Pack1), unload(Desk1, Pack1)`
- `go(Rep, Desk1), unload(Desk1, Rep)`
- `go(Rep, Parking), unload(Parking, Rep)`
- `go(Desk1, Parking), unload(Parking, Desk1)`

In addition to this set, agent `A1` also considers the actions that can be used to fix the faulty functionalities; the set $\mathcal{A}^R$ is:

- `recharge()`
- `refillcoolant()`.

Table 2: Main characteristics of the simulated plans (avg. values) in each scenario.

	SCN3	SCN4	SCN5	SCN6	SCN7	SCN8
# actions	66	70	94	135	180	256
# casual links	223	238	333	387	467	361
# subgoals	46	51	77	73	74	95
# actions per agent	22	18	19	23	26	32
# subgoals per agent	15	13	16	12	11	12

Both actions can be performed only when the agent is located within a parking area. The first one is used to recharge the battery, and hence produces the effect $\mathbf{pwr} = ok$. The second one is used to cool down the engine by refilling the coolant tank, and hence produces the effect $\mathbf{engTmp} = ok$.

Once the actions to be included in Φ have been identified, the agent uses Φ as previously discussed in order to find a conformant plan. Indeed, the agent is able to find two alternative conformant plans. The first one is *Solution 1:*

- `unload(Rep, Pack1)`
- `go(Rep, Parking)`
- `recharge()`
- `refillcoolant()`
- `go(Parking, Rep)`
- `load(Rep, Pack1)`

The alternative *Solution 2* is similar to the previous one, but the two repairing actions are switched. It is worth noting that the first action of the repairing plan undoes the effects achieved by the previous action a_1, but this is necessary in order to allow the agent move even when its functionalities are not in their nominal mode. Then, the agent moves to a parking area where it can perform repairing actions. After this step, the functionalities assumed faulty within the agent diagnosis are now properly working; thus, the agent can resume the execution of the plan. More precisely, the agent moves back to `Rep`, and loads again `Pack1`. At the end of the repairing plan, the agent is now ready to perform again the failed action a_2; this second time, however, the root causes of the failure have been removed so the action is expected to be completed successfully. ■

7 Experimental results.

The proposed control loop has been implemented in Java JDK 1.7. The symbolic formalism of the Ordered Binary Decision Diagrams (OBDDs) [2] has been used to

encode the agents' belief states, and the non-deterministic models of the actions. Monitoring, diagnosis and planning are therefore implemented in terms of standard OBDDs operators (see [3, 7, 10]). Agents are threads running on the same PC [1].

In our experiments we have (software) simulated the office domain introduced in Example 3, in which a team of robotic agents offers a mail delivery service. In particular, as a test case, the environment we used in our experiments consists of 5 office rooms, and involves 2 repositories, 9 desks, 8 doors (connecting two rooms; it is possible to have more doors between the same two rooms), and 12 blocks. All these elements are considered as critical resources, shared by the agents. In addition, there are three parking areas, which are not critical resources, and where agents can perform their repairing actions.

In order to prove the effectiveness of the local recovery strategy, we have considered six alternative scenarios including from 3 to 8 agents. In each scenario we have simulated the execution of 40 MAPs, whose main characteristics are reported in Table 2.

In these six scenarios, we compared the behavior of four strategies when the actual execution of MAPs is affected by the occurrence of faults. More precisely, we randomly injected one fault in each MAP; namely, we randomly selected only one agent, and affected one of its functionalities with a fault. The methodology we have presented easily deals with multiple faults. More precisely, it may be possible to degrade the functionalities of different agents, or even to degrade the functionalities of the same agent more times. In the experimental analysis discussed in this paper, however, we restricted ourselves to a single fault per MAP in order to avoid masking effects that can occur in case of multiple faults.

To test the effectiveness of the repair strategy, each injected fault is repairable, in the sense that there exists a repairing action. However, in some cases an agent cannot recover from a failure because some contextual conditions may prevent the agent from performing the needed repairing actions. For instance, the closest parking area is not reachable by the impaired agent given the current set of resources (doors, in particular) held by the agent itself.

The four strategies we compared are: *no-repair*, the agent in trouble does not handle the failure (the agent just stops the execution of its local plan); *safe-status*, whenever an action failure occurs, the agent in trouble moves into a safe status; *repair*, the agent tries to repair its own plan, when the repair process fails the agent stops the plan execution; *r+s* (repair and safe-status), a combination of the previous scenarios: first the agent tries to repair its plan, in case this step fails the agent reaches a safe status. In the following we also report experimental data under *nominal* conditions (i.e., when no fault occurs), in order to have a benchmark for the different strategies.

Figure 5 shows the average number of actions that have been performed in four strategies. From the picture it emerges that the best strategy is *r+s*. This strategy, in fact, is the most flexible as it can take advantage of both a plan repair, and when this is not possible, of a plan-to-safe-status, which tries to mitigate the impact of an action failure. In Figure 6 we show the number of subgoals actually achieved in the six scenarios by the four strategies. Also in this case strategy *r+s* is the best choice as it

[1] Intel Core i5CPU, 2.53GHz, RAM 4GB, Windows7

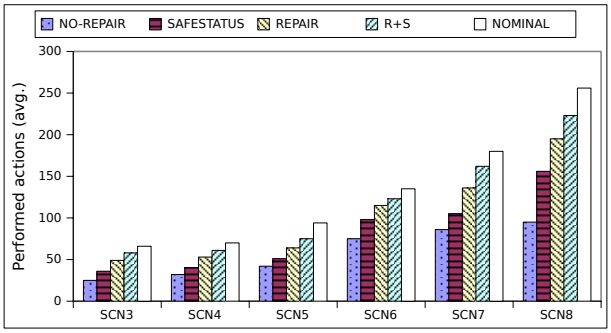

Figure 5: The average number of performed actions in the four repair strategies.

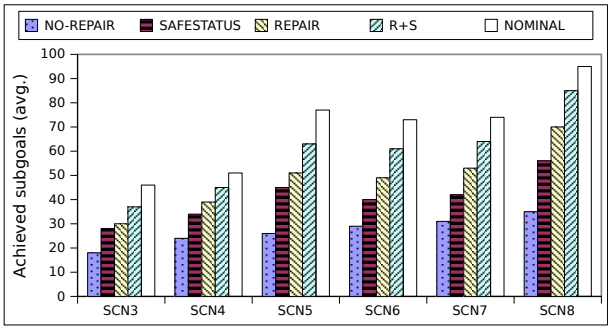

Figure 6: The average number of achieved subgoals in the four repair strategies.

gets the highest number of subgoals.

It is worth noting that the monitoring task is performed very effectively. Indeed, since each agent monitors and diagnoses its own local plan without the need of interacting with others, the monitoring (and diagnosis) computational time is independent of the number of agents in the team. In fact, in all the six scenarios, the CPU time for monitoring a single execution step is in the order of 200 ms., the observed maximum is 302 ms.

The repair phase involving the conformant planner is, instead, slightly more expensive, see Table 3. Of course, strategy *no-repair* is not included as its repair cost is always zero (indeed, this strategy does not attempt any plan repair). The table reports the average CPU time that has been spent for planning either to a repaired state, or to a safe-status.

The *safestatus* strategy can just plan to a safe-status, such a planning problem is in general simpler that a plan repair problem, and hence it is the cheapest strategy, but its effectiveness is limited, as we have already shown.

The *repair* strategy can just plan to a repaired state from which the nominal plan execution can be resumed. Such a planning problem is more complex since it has to restore the nominal conditions in the agent's functionalities, and to satisfy the precon-

Table 3: CPU time [msec] for the repair process.

		repaired	safestatus
SCN3	*r+s*	725 ± 102	937 ± 242
	repair	705 ± 115	-
	safestatus	-	213 ± 56
SCN4	*r+s*	839 ± 151	987 ± 224
	repair	820 ± 127	-
	safestatus	-	198 ± 75
SCN5	*r+s*	878 ± 180	1125 ± 235
	repair	825 ± 137	-
	safestatus	-	219 ± 84
SCN6	*r+s*	925 ± 230	1323 ± 257
	repair	889 ± 221	-
	safestatus	-	247 ± 91
SCN7	*r+s*	1105 ± 234	1402 ± 224
	repair	1078 ± 172	-
	safestatus	-	257 ± 104
SCN8	*r+s*	1356 ± 254	1547 ± 232
	repair	1289 ± 145	-
	safestatus	-	287 ± 127

ditions for the plan segment still to be performed.

The *r+s* strategy can plan both to a repaired state and to a safe-status. Indeed, when a plan to a repaired state exists, the strategy behaves similarly to the *repair* strategy. However, when such a planning step fails, the *r+s* strategy tries a plan to safe-status, in these cases the computational cost of *r+s* is in the order of one second or more. It is worth noting, however, that even though *r+s* can invoke, at least in some cases, the conformant planner twice, the computational cost is not doubled. This happens because in most of the cases *r+s* detects very early that a conformant plan to a repaired state does not exist, and hence the overhead introduced by this planning step is still acceptable.

The computational cost of the conformant planner is mainly due to the dimensions of the OBDDs it has to handle. More precisely, while the monitoring phase handles just one action model, whose average size is 364 nodes, the Φ macro-operator involves 40 actions, on average, and its size is 4876 nodes. The OBDD encoding Φ is then used at each step of the conformant algorithm to increase the lengths of the current plans. In particular, the found repairing plans include, on average, 8 actions.

8 Conclusions

In this paper we have formalized a closed loop of control involving three main activities: plan execution monitoring, agent diagnosis and plan repair.

The paper contributes to show the importance of a repair strategy driven by a fail-

ure analysis which highlight the root causes of an action failure. Depending on the (possibly multiple) faults and on the activities of the agent in trouble, different course of actions are synthesized either for recovering the action failure (if the local repairing plan exists), or to bring the agent in a safe status and limit the impact of the failure.

The experimental results show that the proposed methodology is adequate to promptly react to an action failure and to actually mitigate the harmful effects of the failure. Also the computational cost of the approach is affordable since the search for a recovery plan is strongly constrained by the agent diagnosis.

The proposed framework can be extended to deal with more sophisticated notions multi-agent plan. First of all, concurrency constraints can be introduced to model joint actions (see e.g., [11]). A more interesting extension concerns the temporal dimension (see e.g., [13, 14]). Dealing with temporal plans has a strong impact on the conformant planner, which has to find a repairing plan that meets the set of missing goals, and that can be executed without violating any temporal constraint.

References

[1] L. Birnbaum, G. Collins, M. Freed, and B. Krulwich. Model-based diagnosis of planning failures. In *Proc. AAAI90*, pages 318–323, 1990.

[2] R. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computer Surveys*, 24:293–318, 1992.

[3] A. Cimatti and M. Roveri. Conformant planning via symbolic model checking. *JAIR*, 13:305–338, 2000.

[4] J. S. Cox, E. H. Durfee, and T. Bartold. A distributed framework for solving the multiagent plan coordination problem. In *Proc. AAMAS05*, pages 821–827, 2005.

[5] K. Decker and J. Li. Coordinating mutually exclusive resources using GPGP. *Journal of AAMAS*, 3(2):113–157, 2000.

[6] B. Horling and V. Benyo, B. Lesser. Using self-diagnosis to adapt organizational structures. In *Proc. ICAA'01*, pages 529–536, 2001.

[7] R. M. Jensen and M. M. Veloso. Obdd-based universal planning for synchronized agents in non-deterministic domains. *JAIR*, 13:189–226, 2000.

[8] M. Kalech and G. A. Kaminka. On the design of coordination diagnosis algorithms for teams of situated agents. *AI*, 171:491–513, 2007.

[9] M. Kalech and G.A. Kaminka. On the design of social diagnosis algorithms for multi-agent teams. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI03)*, pages 370–375, 2003.

[10] R. Micalizio. Action failure recovery via model-based diagnosis and conformant planning. *Computational Intelligence*, 29(2):233–280, 2013.

[11] R. Micalizio and P. Torasso. Monitoring the execution of a multi-agent plan: Dealing with partial observability. In *Proc. of ECAI'08*, pages 408–412, 2008.

[12] R. Micalizio and P. Torasso. Agent cooperation for monitoring and diagnosing a map. In *MATES*, volume 5774 of *Lecture Notes in Computer Science*, pages 66–78, 2009.

[13] Roberto Micalizio and Gianluca Torta. Diagnosing delays in multi-agent plans execution. In *ECAI*, pages 594–599, 2012.

[14] Roberto Micalizio and Gianluca Torta. Diagnosing dependent action delays in temporal multiagent plans. In *Research and Development in Intelligent Systems XXX, Incorporating Applications and Innovations in Intelligent Systems XXI Proceedings of AI-2013*, pages 157–171, 2013.

[15] N. Roos and C. Witteveen. Models and methods for plan diagnosis. *Journal of Autonomous Agent and Multiagent Systems*, 19(1):30–52, 2009.

[16] G. Steinbauer and F. Wotawa. Enhancing plan execution in dynamic domains using model-based reasoning. *LNAI*, 5314:510–519, 2008.

[17] R. van der Krogt and de Weerdt. Plan repair as an extension of planning. In *Proc. of ICAPS'05*, pages 284–259, 2005.

[18] Daniel S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27–61, 1994.